Script **generated by TTT**

Title: Petter: Compiler Construction (18.06.2020) - 34: Attribute Grammar Notation

Date: Thu Jun 18 09:36:15 CEST 2020

Duration: 10:17 min

Pages: 6

Implementation Strategy

- attach an attribute empty to every node of the syntax tree
- compute the attributes in a *depth-first* post-order traversal:
  - at a leaf, we can compute the value of empty without considering other nodes
  - the attribute of an inner node only depends on the attribute of its children
- the empty attribute is a *synthesized* attribute

Implementation Strategy

- attach an attribute empty to every node of the syntax tree
- compute the attributes in a *depth-first* post-order traversal:
  - at a leaf, we can compute the value of empty without considering other nodes
  - the attribute of an inner node only depends on the attribute of its children
- the empty attribute is a *synthesized* attribute

in general:

Definition

An attribute at N is called

- *inherited* if its value is defined in terms of attributes of N's parent, siblings and/or N itself (root $\hookrightarrow$ leaves)
- *synthesized* if its value is defined in terms of attributes of N's children and/or N itself (leaves $\rightarrow$ root)

Example: Attribute Equations for empty

In order to compute an attribute *locally*, specify attribute equations for each node

Example: Attribute Equations for empty

In order to compute an attribute *locally*, specify attribute equations for each node depending on the *type* of the node:

In the Example from earlier, we did that intuitively:

for leaves: $r \equiv$  we define $\mathsf{empty}[r] = (x \equiv \epsilon)$.

otherwise:

$$\begin{aligned} \mathsf{empty}[r_1 \mid r_2] &= \mathsf{empty}[r_1] \vee \mathsf{empty}[r_2] \\ \mathsf{empty}[r_1 \cdot r_2] &= \mathsf{empty}[r_1] \wedge \mathsf{empty}[r_2] \\ \mathsf{empty}[r_1^*] &= t \\ \mathsf{empty}[r_1?] &= t \end{aligned}$$

Specification of General Attribute Systems

General Attribute Systems

In general, for establishing attribute systems we need a flexible way to *refer to parents and children*:

$\rightsquigarrow$ We use consecutive indices to refer to neighbouring attributes

$\mathsf{attribute}_k[0]$: the attribute of the current root node
$\mathsf{attribute}_k[i]$: the attribute of the i-th child $(i > 0)$

Specification of General Attribute Systems

General Attribute Systems

In general, for establishing attribute systems we need a flexible way to *refer to parents and children*:

$\rightsquigarrow$ We use consecutive indices to refer to neighbouring attributes

$\mathsf{attribute}_k[0]$: the attribute of the current root node
$\mathsf{attribute}_k[i]$: the attribute of the i-th child $(i > 0)$

... the example, now in general formalization: